Prednosti i nedostaci Redis tehnologije u web aplikacijama

Bernatović, Ivan

Master's thesis / Diplomski rad

2018

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **Josip Juraj Strossmayer University of Osijek, Faculty of Electrical Engineering, Computer Science and Information Technology Osijek / Sveučilište Josipa Jurja Strossmayera u Osijeku, Fakultet elektrotehnike, računarstva i informacijskih tehnologija Osijek**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:200:964884

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2024-03-29**

Repository / Repozitorij:

Faculty of Electrical Engineering, Computer Science and Information Technology Osijek

zir.nsk.hr

Image not found or type unknown

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU

FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA

Sveučilišni studij

PREDNOSTI I NEDOSTACI REDIS TEHNOLOGIJE U WEB APLIKACIJAMA

Diplomski rad

Ivan Bernatović

Osijek, 2018.

Sadržaj

1. UVOD 1

2. PROGRAMSKA PODRŠKA I SKLOPOVLJE 2

2.1 Operacijski sustav 2

2.2 HTTP poslužitelj 2

2.3 Sustav za upravljanje relacijskim bazama podataka 2

2.4 PHP programski jezik 3

2.5 Redis baza podataka kao pričuvna memorija 3

2.6 Testno sklopovlje 4

2.7 Sažetak konfiguracije programske podrške 5

3. TESTNA APLIKACIJA, KONFIGURACIJA I PODACI 7

3.1 Opis testne aplikacije 7

3.2 Sažetak tehničkih specifikacija 8

3.3 Korištenje Redis baze podataka 8

3.4 Korištenje Redisa u PHP aplikacijama 10

3.5 Redis kao upravljački program pričuvne memorije u Laravel aplikacijama 11

3.6 Dohvaćanje i spremanje podataka za naslovnicu u pričuvnu memoriju 13

3.7 Korisnički podaci 15

4. IZVOĐENJE TESTOVA 17

4.1 Ciljevi testiranja 17

4.2 Programska podrška za testiranje 17

4.3 Definiranje testnog plana i podataka 18

4.4 Izvedba testa opterećenja do prve pogreške 19

4.5 Izvedba testa neprekidnog opterećenja bez pogrešaka kroz određeni period 20

5. ZAKLJUČAK 22

LITERATURA 23

SAŽETAK 24
ABSTRACT 25
ŽIVOTOPIS 26
PRILOG 27

1. UVOD

Web aplikacije u 21. stoljeću postaju sve kompleksnije, dostupne su većem broju korisnika i procesiraju sve više podataka. U skladu s tim javlja se potreba za boljim performansama. Postoje razne tehnike za optimizaciju performansi, od poboljšanja kvalitete i učinkovitosti koda do korištenja većeg broja jačeg sklopovlja. Međutim, ponekad zbog financijskih ili logističkih razloga nije moguće daljnje poboljšavati kvalitetu koda ili ulagati više novaca u bolje sklopovlje. Jedna od vrlo učinkovitih tehnika poboljšanja performansi web aplikacije je intenzivnije korištenje pričuvne memorije.

Zadatak ovoga rada je demonstrirati kako implementirati Redis bazu podataka u PHP web aplikaciju te prikazati utjecaje na performanse aplikacije. U ovome radu Redis će se koristiti kao baza podataka pričuvne memorije pohranjena u radnoj memoriji. Očekivani krajnji rezultati su bolje performanse aplikacije odnosno mogućnost posluživanja većeg broja klijentskih zahtjeva nego prije. Redis se često naknadno implementira u LEMP stogove jer LEMP postava nema strogo definiran način spremanja pričuvne memorije. Pričuvna ili *cache* memorija je vrlo općenit pojam i njezino jasno definiranje najviše ovisi o kontekstu. Postoji jako puno vrsta pričuvnih memorija, može ih se naći u procesorskim jedinicama, HTTP poslužiteljima, web aplikacijama, internetskim preglednicima itd. No neovisno o kontekstu, glavna svrha pričuvne memorije je brz pristup podacima koji se često koriste, a inače nisu. Za potrebe rada izrađena je testna aplikacija koju se koristi za primjere implementacije Redisa te testiranje performansi.

U prvom djelu rada opisana je korištena programska podrška i sklopovlje. Drugi dio sastoji se od opisa testne aplikacije te primjera kako koristiti Redis u PHP aplikacijama, sa i bez dodatnih biblioteka. Treći dio rada opisuje programsku podršku za testiranje performansi aplikacije, opisuje testnu metodologiju te rezultate testova.

2. PROGRAMSKA PODRŠKA I SKLOPOVLJE

Svi testovi i primjeri izvodit će se na LEMP softverskom stogu (eng. *stack*). LEMP se sastoji od kombinacije Linux/GNU distribucije, Nginx HTTP poslužitelja, MariaDB sustava za upravljanje relacijskim bazama podataka te PHP programskog jezika [1]. LEMP je često korištena postava i postoji puno dokumentacije kako instalirati i konfigurirati poslužitelj baziran na LEMP-u. LEMP predstavlja samo ključne komponente stoga nije strogo definiran i ostavlja prostor za dodavanje raznih drugih komponenti kao što je Redis u slučaju ovoga rada.

2.1 Operacijski sustav

LEMP ne definira strogo koji operacijski sustav se treba koristiti ali pretpostavlja da je sustav baziran na Linux jezgri. Zbog jednostavnosti, popularnosti i znanja autora ovoga rada kao operacijski sustav koristi se Ubuntu Server 18.04 LTS. Ubuntu je najpopularnija desktop Linux/GNU distribucija koja ima i svoju poslužiteljsku verziju. Razlika između *desktop* i poslužiteljske verzije je što verzija za poslužitelje ne sadržava ključne komponente koje su vezane za grafičko korisničko sučelje (eng. *graphical user interface* - GUI) te programe koji podrazumijevaju prisutnost sustava za prozore (eng. *windowing system*). Ubuntu 18.04 je objavljen u travnju 2018. godine i u trenutku pisanja i objavljivanja ovoga rada je to aktualna verzija s dugoročnom podrškom. Glavna svrha LTS verzija je stabilnost softvera. Najčešće je u takvim verzijama prisutan vrlo dobro testiran i pouzdan ali ne i najnoviji softver. Time se garantira visoki *uptime* poslužitelja i izbjegavaju se potencijalne greške i problemi. Više o shemi verzioniranja Ubuntu-a može se pročitati na službenoj stranici.

2.2 HTTP poslužitelj

Kao HTTP poslužitelj u LEMP stogu koristi se Nginx HTTP Server (u ostatku teksta koristit će se samo skraćeno Nginx). Nginx je jedan od najpopularnijih HTTP poslužitelja i dostupan je na UNIX, Linux i Windows platformama. Trenutačno je najnovija stabilna grana 1.14.x [2]. Budući da je prva verzija Nginx-a objavljena 2004. godine, Nginx ima bogat ekosustav raznih proširenja i vrlo dobru službenu i neslužbenu dokumentaciju. Nginx je također i vrlo konfigurabilan i zbog toga je vrlo popularan izbor za puno slučajeva korištenja.

2.3 Sustav za upravljanje relacijskim bazama podataka

Kao sustav za upravljanje relacijskim bazama podataka (eng. *relational database management system – RDBMS*) u LEMP-u stogu koristi se MySQL ili MySQL kompatibilna

tehnologija kao što je MariaDB. MariaDB projekt je nastao kao reakcija Oracle-ove kupnje MySQL jer su developeri postali zabrinuti potencijalnim „zatvaranjem“ izvornog koda. Stoga je dio developera preuzeo trenutačnu verziju MySQL izvornog koda i nastavio ju razvijati pod imenom MariaDB. Primarni fokus je održati kompatibilnost s originalom – MySQL-om i ostati softver otvorenog koda [3]. Kao i većina drugih RMDBS-ova, i MySQL podržava korištenje SQL-a (skraćeno od *structurual query language*, hrv. strukturirani upitni jezik). MySQL je iznimno popularan RDBMS i koristi se u mnogim popularnim i visoko prometnim aplikacijama kao što su Facebook, YouTube, Twitter i mnogi drugi. U trenutku pisanja ovoga rada najnovija stabilna grana je 5.7.x. Iako je MySQL najpopularnija implementacija SQL-a, za potrebe testiranja ovoga diplomskog rada koristit će se MariaDB 1.13.

2.4 PHP programski jezik

PHP programski jezik je skriptni jezik primarno dizajniran za dinamičke web aplikacije. PHP je inicijalno bila kratica za *Personal Home Page*, a kasnije je dobilo novo značenje u obliku rekurzivnog akronima: *PHP: Hypertext Pre-processor* [4]. Međutim, zbog visoke fleksibilnosti može ga se koristiti i kao programski jezik opće namjene. PHP kod se uobičajeno izvršava pomoću PHP interpretera koji je implementiran kao dodatni modul HTTP servera ili pomoću CGI (eng. *Common Gateway Interface*) binarne datoteke koja isto može biti dio HTTP servera kao što su Apache ili Nginx [5]. PHP ima veliku povijest promjena i ažuriran je mnogo puta. Podržava paradigme i strukturnog i objektno orijentiranog programiranja. U PHP je ugrađeno puno funkcija koje olakšavaju programiranje web aplikacije. Npr. ima puno funkcija za manipuliranje nizovima, za parsiranje raznih tipova podataka itd. Budući da je iznimno popularan programski jezik postoji puno PHP modula koji dodatno proširuju funkcionalnosti na binarnoj razini kao što su napredne enkripcije i napredno manipuliranje slika. Koriste ga mnoge velike tvrtke i najveće aplikacije na svijetu kao što su Facebook, Wikipedia, Slack itd. U trenutku pisanja ovoga rada aktualna stabilna grana PHP-a je 7.2.x.

2.5 Redis baza podataka kao pričuvna memorija

Redis je baza podataka otvorenog koda koja funkcionira po principu „ključ-vrijednost“ i svoje podatke čuva u radnoj memoriji. Redis podržava spremanje raznovrsnih apstraktnih tipova podataka kao što su nizovi znakova, popisi, mape, setovi, sortirani setovi, *bitmap* itd. Zbog principa spremanja „ključ-vrijednost“ Redis spada u NoSQL (skraćeno od eng. izraza *Not Only SL*, hrv. „Ne samo SQL“) kategoriju baza podataka. Glavna značajka NoSQL baza podataka je to

što svoje podatke ne sprema isključivo u tabličnom obliku s relacijama kao što je to slučaj kod relacijskih baza podataka.

U mnogim slučajevima je implementacija tehnologije poput Redisa jedina opcija kojom se može podnijeti visoki promet cijele infrastrukture. U takvim slučajevima ne može pomoći čak ni horizontalno skaliranje – dodavanje dodatnih poslužitelja u *backend* infrastrukturu, a ni vertikalno skaliranje – povećavanje snage pojedinačnog poslužitelja ugradnjom snažnijih komponenti koje direktno utječu na brzinu izvođenja. *Backend* infrastrukturu čine svi poslužitelji, njihove namjene, zadaće i relacije te specifična konfiguracija pojedinačnog poslužitelja. Svaka tvrtka koja ima svoju *backend* infrastrukturu bi trebala imati dokumentaciju i zaposlenike čija je glavna zadaća održavanje i optimizacija infrastrukture.. Pristup podacima u Redisu je iznimno brz jer se svi podaci spremaju u radnu memoriju računala. Također je i vrlo konfigurabilan i vrlo se dobro skalira jer ima podršku za grozdove, odnosno moguće je imati više Redis instanci u *backend* infrastrukturi koje se koriste na više različitih poslužitelja, a pri tome je pristup tim podacima i dalje jednostavan i nema puno dodatnog konfiguriranja pri odlučivanju koji podaci će se pohranjivati u kojoj instanci.

2.6 Testno sklopovlje

Kao testno sklopovlje koristit će se *Virtual Private Server* (skraćeno VPS) instanca od davatelja usluge tvrtke DigitalOcean. Glavni razlog odabira VPS tehnologije umjesto fizičkih poslužitelja je veća fleksibilnost pri mijenjanju konfiguracije te manji troškovi. Za razliku od fizičkih poslužitelja, VPS instance su izolirani virtualni strojevi koji su dio velike mreže superračunala pogonjene *hypervisor* programskom podrškom. Njima se pristupa jednako kao i fizičkom poslužitelju, na njima je u potpunosti instaliran operacijski sustav po izboru i u kontekstu programske podrške je jednako fleksibilan kao i fizički poslužitelj. Budući da su VPS instance virtualni strojevi, u slučaju potrebe za većim brojem instanci može se vrlo brzo instancirati nove virtualne poslužitelje. U većini slučajeva su VPS instance poželjnije od fizičkih poslužitelja. Međutim, ako je korisniku poslužitelja potrebno posebno sklopovlje ili bolje performanse određenih komponenti onda je fizički poslužitelj bolji izbor jer pruža veću fleksibilnost u kontekstu sklopovlja.

DigitalOcean svoje instance virtualnih poslužitelja stilizirano naziva *Droplet*. *Dropleti* su dostupni u mnogobrojnim predefiniranim konfiguracijama koje su određene brojem virtualnih procesorskih jezgri, količinom radne memorije, količinom prostora na SSD-u, dopuštenim mjesečnim podatkovnim prometom te cijeni. Također su dostupni *Dropleti* koji imaju CPU-ove bolji

performansi od standardnih *Dropleta* što je korisno za poslužitelje koji izvršavaju operacije koje intenzivno koriste CPU jezgre. DigitalOcean naplaćuje korištenje *Dropleta* po satu.

Za potrebe testiranja koristit će se *Droplet* sljedeće konfiguracije:

- 2 virtualne procesorske jezgre
- 4 GB radne memorije
- 80 GB prostora na disku (SSD visokih performansi)
- 4 TB podatkovnog prometa
- 20 USD mjesečno, 0,03 USD po satu (127,24 HRK mjesečno, 0,19 HRK po satu)

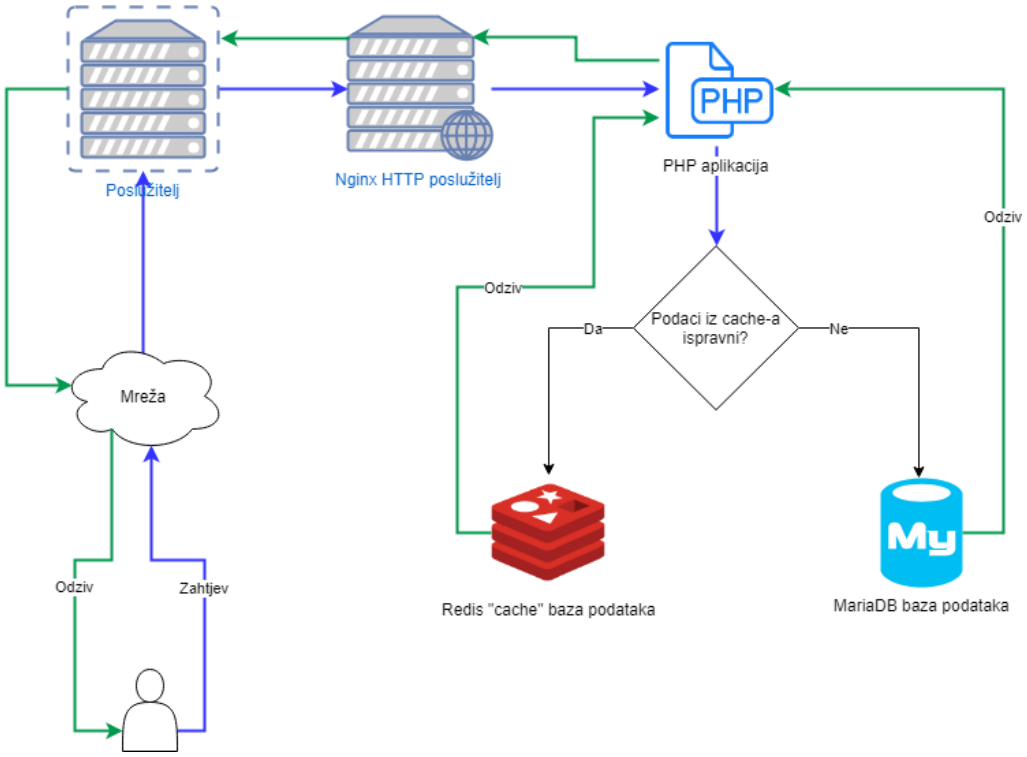
2.7 Sažetak konfiguracije programske podrške

U sljedećoj tablici su prikazane ključne komponente programske podrške testnog okruženja.

Tab. 2.1. *Ključna korištena programska podrška*

Naziv	**Verzija**	**Svrha**
Ubuntu	18.04 LTS	Operacijski sustav
Nginx	1.14	HTTP poslužitelj
PHP	7.2.9	Jezik za procesiranje logike aplikacije
MariaDB	1.3.9	RDBMS
Redis	4.0.8	Baza podataka za pričuvnu memoriju

Općeniti prikaz sustava je demonstriran na slici 1.1. Korisnik preko mreže pošalje HTTP zahtjev poslužitelju koji je konfiguriran da zahtjev proslijedi PHP aplikaciji. PHP aplikacija procesira zahtjev i po potrebi će joj trebati određeni podaci. U PHP aplikaciji mora postojati logika koja provjerava jesu li podaci u Redis bazi podataka ili je potrebno nove podatke dohvatiti iz MariaDB baze podataka. Nakon što se to utvrdi PHP aplikacija dohvati podatke, generira odziv koji je najčešće u HTML ili JSON obliku te odziv vrati HTTP poslužitelju. Poslužitelj preko mreže odziv vrati klijentu koji u svojem internetskom pregledniku vidi web stranicu.



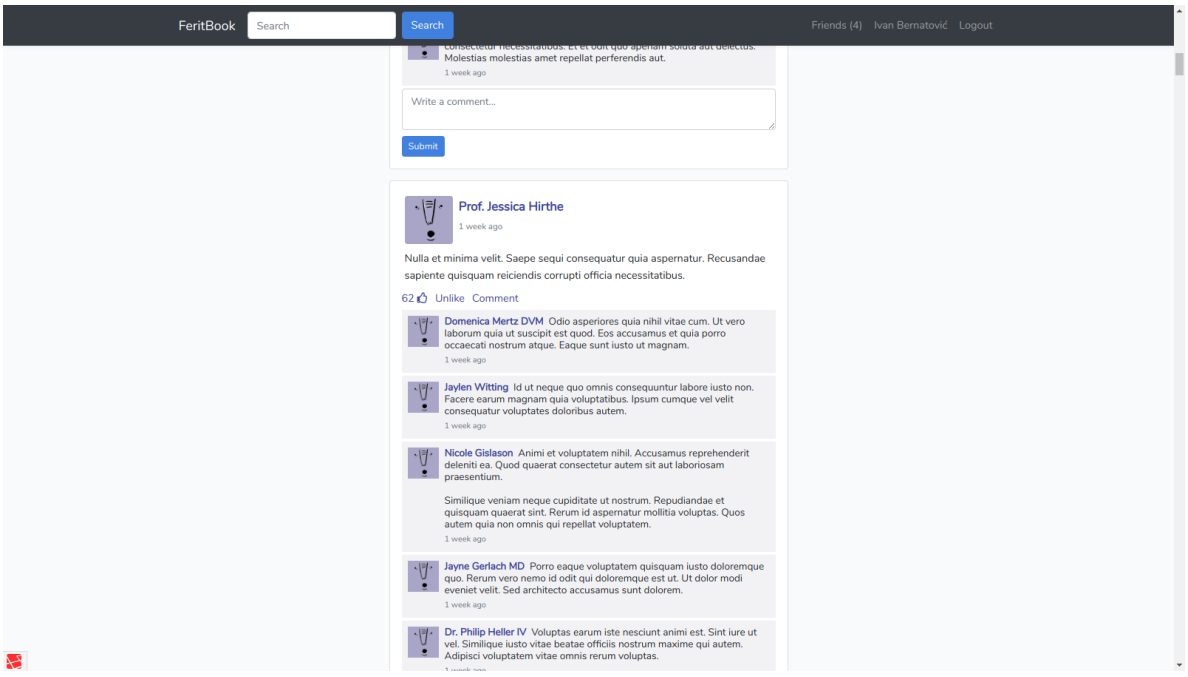
Sl. 2.1. *Prikaz osnovne arhitekture programske podrške*

3. TESTNA APLIKACIJA, KONFIGURACIJA I PODACI

3.1 Opis testne aplikacije

Za potrebe ovog rada i testiranja izrađena je posebna testna aplikacija. Radi se o društvenoj mreži nazvanoj Feritbook koja ima nekoliko osnovnih funkcionalnosti koje se očekuju od društvenih mreža:

- pretraga korisnika
- dodavanje i uklanjanje korisnika kao prijatelja
- uređivanje osnovnih korisničkih podataka i slike profila
- objavljivanje statusa i fotografija
- pregled korisničkih profila i objava prijatelja
- pregled svih objava prijatelja na naslovnici poredanih od najnovijeg prema najstarijem
- mogućnost komentiranja i sviđanja objave (status ili fotografija)



Sl. 3.1. *Prikaz naslovnice testne aplikacije*

Izvorni kod aplikacije dostupan je kao Git repozitorij posluživan na sljedećoj GitLab stranici: https://gitlab.com/IvanBernatovic/feritbook.

3.2 Sažetak tehničkih specifikacija

Testna aplikacije je napravljena korištenjem Laravel radnog okvira (eng. *framework*). Laravel je popularni framework napisan u PHP programskom jeziku. Laravel je implementacija *Model-View-Controller* (skraćeno MVC) paradigme u kontekstu web aplikacije. Glavna svrha MVC paradigme je jasna podjela komponenti po svojoj svrsi:

- *Model* je tip komponente koja sadrži poslovnu logiku programa
- *View* je tip komponente koja sadrži prezentacijsku logiku programa
- *Controller* je tip komponente koja rukovodi korisničkim inputima i prosljeđuje logiku modelima

MVC paradigma je općenita paradigma u računarstvu koja nije implementirana samo u kontekstu web aplikacija nego i u drugim oblicima [6]. Laravel osim što implementira MVC paradigmu također pruža mnogo korisnih alata. Glavni aduti tog radnog okvira su visoka produktivnost, čitkost koda i lakoća korištenja. Zbog tih značajki Laravel omogućuje ubrzani razvoj aplikacije i brzo iteriranje što ga posebno čini popularnim u manjim timovima. U ovoj testnoj aplikaciji korištena je verzija 5.6 što je u trenutku izrade bila najažurnija verzija.

3.3 Korištenje Redis baze podataka

Redis je baza podataka otvorenog izvornog koda koja podržava spremanje podataka direktno u radnu memoriju. Također je moguće i trajno spremanje podatke na disk ali su u tome slučaju performanse lošije. Podržano je mnogo tipova podataka kao što su nizovi znakova, asocijativni nizovi, popisi, setovi, sortirani setovi, bitmap datoteke itd. [7]. Također Redis ima neke korisne značajke kao što su:

- replikacija po principu *master-slave* paradigme
- skriptiranje pomoću Lua programskog jezika
- transakcije (izvršavanje grupe naredbi u jednom koraku)
- sustav za nadgledanje, konfiguriranje i auto inicijalizaciju instanci Redisa zvan Sentinel
- mogućnost postavljanja grozdova distribuiranih na mnogobrojnim poslužiteljima

Da bi poslužitelj mogao koristiti Redis potrebno je instalirati dvije osnovne komponente Redisa koje su odvojene radi transparentnosti, a to su:

1. Redis server - sama instanca Redisa zadužena za pohranu i procesiranje naredbi

2. Redis klijent - program pomoću kojeg se komunicira s Redis serverom; korišten direktno od strane korisnike ili od strane neke druge aplikacije

Kako bi koristili Redis potrebno je pokrenuti program *redis-cli* u ljusci operacijskog sustava. Nakon toga moguće je izvršavati naredbe. Jedan od osnovnih slučajeva korištenja Redisa je pohrana podataka. Kao što je već rečeno u prethodnim poglavljima, Redis omogućuje spremanje podataka u obliku ključ-vrijednost. To znači da svaka spremljena vrijednost ima svoj ključ putem kojeg je moguće dohvatiti tu vrijednost. Primjerice, u slučaju potrebe pohrane vrijednosti niza znakova “bar”, potrebno je koristiti naredbu SET u sljedećem obliku:

SET {naziv ključa} {vrijednost}

Stoga ako želimo pohraniti vrijednost „bar” pod ključem „foo”, koristit ćemo sljedeću naredbu:

SET foo 'bar'

Za dohvaćanje vrijednosti potrebno je znati ključ pod kojim je ta vrijednost spremljena te je potrebno koristiti naredbu GET. GET naredba koristi se na sljedeći način:

GET {naziv ključa}

Konkretno ako želimo dohvatiti vrijednost pod ključem „foo“, potrebno je izvršiti sljedeću naredbu:

GET foo

Na ekranu će biti ispisana vrijednost 'bar'.

Budući da Redis podržava nekoliko tipova podataka dostupne su naredbe koje specifično i isključivo rade samo s određenim tipom. Primjerice, naredbe INCR {ključ} i DECR {ključ} moguće je koristit samo ako je vrijednost spremljena pod tim ključem cijeli broj. Razlog tome je što te naredbe inkrementiraju ili dekrementiraju vrijednosti stoga ih nije moguće primijeniti na npr. niz znakova ili na set. Svaki pokušaj korištenja tih naredbi za vrijednosti drugih tipova rezultirat će pogreškom. Za potrebe ovog diplomskog rada koristit će se samo SET i GET naredbe u koje će se pohranjivati podaci serijalizirani kao niz znakova. Razlog tome je što je testnoj aplikaciji isključivo potrebna brzina pristupa podacima, a ne složena manipulacija nekih tipova podataka ili neke druge operacije koje Redis podržava.

3.4 Korištenje Redisa u PHP aplikacijama

Nakon što je Redis instaliran moguće ga je koristiti u drugim aplikacijama. Teoretski ga može koristiti bilo koja aplikacija koja ima pristup ljusci operacijskog sustava ili izvršavanju drugih programa na neki drugi način. Najosnovniji način korištenja Redisa u PHP aplikaciji je pomoću funkcije za ostvarivanje mrežne veze putem *socketa* zvane *fsockopen*. Ta funkcija kao povratnu vrijednost vraća tzv. *handle*. *Handle* je PHP tip podatka koji predstavlja pokazivač na neki resurs sustava nad kojim je moguće izvršavati određene operacije kao npr. *fwrite* funkciju [8].

```
<?php

// otvaranje veze gdje varijabla $redisClient predstavlja handle
putem kojeg se mogu izvršavati određene funkcije
$redisClient = fsockopen('127.0.0.1', 6379, $errCode, $errStr);

$rawCommand = "*2\r\n\$4\r\nEcho\r\n\$12\r\nhello world!\r\n";

// izvršavanje naredbe
fwrite($redisClient, $rawCommand);

$rawResponse = fgets($redisClient);
echo $rawResponse; // $12

$rawResponse = fgets($redisClient);
echo $rawResponse; // hello world!
```

Primjer koda 3.1. *Komunikacija s Redisom u PHP kodu*

Iako je sasvim moguće koristit Redis na ovakav način u PHP aplikacijama, postoje i pouzdanije metode kako koristiti Redis. Budući da je Redis vrlo popularan program popularna je i podrška programera i programerskih zajednica koje nisu dio Redis organizacije. Rezultat toga su mnogobrojne biblioteke za mnoge programske jezike, platforme i okoline. Postoji nekoliko popularnih biblioteka za korištenje Redisa u PHP-u, a jedna od njih naziva se Predis.

Predis je PHP biblioteka otvorenog koda razvijena od strane PHP programerske zajednice. Predis omogućuje jednostavno korištenje Redisa unutar PHP koda. Podržava spajanje na Redis klijent i izvršavanje mnogih Redis operacija. Primjerice. proces ostvarivanja veze, postavljanja vrijednosti i dohvaćanja vrijednosti pomoću ključa može se svesti na samo tri linije koda, kao što je demonstrirano u sljedećem primjeru [9]:

```
<?php

$client = new Predis\Client();
$client->set('foo', 'bar');
$value = $client->get('foo');
```

Primjer koda 3.2. *Prikaz korištenja Predis biblioteke*

Predis omogućuje izvršavanje gotovo svih operacija koje Redis nudi, uključujući transakcije, korištenje Lua jezike te grozdove. Budući da te druge operacije nisu korištene u testnoj aplikaciji one neće biti detaljnije opisane. Budući da je Predis biblioteka otvorenog koda, sam izvorni kod dostupan je kao Git repozitorij posluživan na sljedećoj GitHub stranici: https://github.com/nrk/predis. Na toj stranici moguće je pregledati izvorni kod biblioteke ali i osnovnu dokumentacija gdje se nalaze upute za instalaciju te korištenje biblioteke.

3.5 Redis kao upravljački program pričuvne memorije u Laravel aplikacijama

Budući da je Laravel moderni radni okvir fokusiran na produktivnost dostupni su razni alati pa tako i pojednostavljeno upravljanje pričuvnom memorijom. Jedan od glavnih ciljeva Laravel-a je imati unificirani API (eng. *Application programmable interface*) za neke temeljne funkcionalnosti aplikacije kao što su pristup i korištenje baza podataka, *sessioni*, upravljanje mailovima, upravljanje pričuvnom memorijom, upravljanje datotečnim sustavom itd. Svaka od tih temeljnih funkcionalnosti ima svoju osnovnu konfiguraciju koju je moguće promijeniti ali i u potpunosti zamijeniti nečim drugim.

Primjerice, Laravel ima klasu *Illuminate\Support\Facades\Mail* koja omogućuje slanje maila korištenjem funkcije *send*. Ta klasa i funkcija se uvijek jednako koriste, neovisno na koji način je zapravo mail poslan. Moguće je konfigurirati Laravel da koristi mail poslužitelj koji je instaliran na istom poslužitelju kao i Laravel instalacija. Također je moguće koristiti neki drugi servis za slanje maila (koji koristi HTTP protokol za uspostavljanje veze) ili neki drugi mail poslužitelj po izboru. Međutim, korištenje Mail klase je uvijek isto jer je konfiguracija odvojena od tih klasa - ona je visoka apstrakcija nad temeljnim pokretačem tih operacija - mail driver-om [10].

U skladu s tim Laravel ima apstrakciju i nad upravljanjem pričuvnom memorijom. Za upravljanje pričuvnom memorijom koristi se klasa *Illuminate\Support\Facades\Cache*. Ta klasa pruža API za jednostavno postavljanje i čitanje podataka iz pričuvne memorije. Također omogućuje i neke druge operacije vezane za potpuno brisanje pričuvne memorije ili samo nekih pojedinačnih dijelova (direktno brisanje po ključu ili brisanje više jedinica koristeći tzv. Oznake pričuvne memorije). U Laravel-u su dostupna 4 upravljačka programa pričuvne memorije, a to su [11]:

1. Datotečni - upravlja pričuvnom memorijom koristeći datotečni sustav u obliku datoteka
2. Baza podataka – upravlja pričuvnom memorijom koristeći zadanu bazu podatka
3. Redis - upravlja pričuvnom memorijom koristeći Redis bazu podataka
4. Memcached - upravlja pričuvnom memorijom koristeći Memcached sustav za pričuvnu memoriju

Po zadanim postavkama Laravel koristi datotečni *driver*. *Driver* za upravljačku memoriju može se promijeniti u Laravel-ovoj glavnoj datoteci za postavljanje osnovnih varijable okoline - *.env*. Ovisno o izboru *drivera*, potrebno je instalirati dodatne biblioteke [11]. Primjerice za Redis je potrebno instalirati ranije spomenutu PHP biblioteku Predis, a za Memcached je potrebno instalirati posebnu PHP ekstenziju. Pojedinosti svakog *drivera* mogu se konfigurirati u konfiguracijskoj datoteci *config/cache.php*. Treba naglasiti da nemaju svi *driveri* iste mogućnosti stoga nisu sve funkcije dostupne za sve *drivere*. Većina funkcija je dostupna za Redis i Memcached *drivere*, dok je datotečni *driver* ima najviše ograničenja jer je trivijalan u odnosu na dva spomenuta *drivera*. U slučaju potrebe za nekim drugim *driverom* koji bi koristio neki drugi program kao bazu podataka za pričuvnu memoriju, Laravel omogućuje i kreiranje vlastitih *drivera*. U tom slučaju potrebno je dobro razumjeti taj *driver* i potrebno je implementirati nužne Laravel *interface-e* za upravljanje pričuvnom memorijom.

Jedne od osnovnih operacija dostupnih u *Cache* klasi su spremanje i čitanje podataka iz pričuvne memorije. Također je moguće i specificirati vrijeme trajanja tog podatka nakon kojeg se on briše iz pričuvne memorije. *Cache* klasa također omogućuje i inkrementiranje i dekrementiranje podataka koji su spominjani u ranijim poglavljima. Načini korištenja *Cache* klase za postavljanje i dohvaćanje demonstrirani su na sljedeći način:

```
<?php

// trajno sprema vrijednost "bar" pod ključem "foo"
Cache::forever("foo", "bar");

// čita vrijednost ključa "foo" što je niz znakova "bar" te ga sprema
u varijablu $baz
$baz = Cache::get("foo");
```

Primjer koda 3.3. *Korištenje Laravel Cache klase za pohranu i čitanje podataka iz pričuvne memorije*

Većina *drivera* za pričuvnu memoriju (svi osim datotečnog) omogućuju korištenje oznaka. Oznake su dodatna metoda za upravljanje jer omogućuju lakše grupiranje pričuvne memorije. Prilikom

pohrane podataka u pričuvnu memoriju moguće je odrediti jednu ili više oznaka za ključ. Više ključeva može imati jednu ili više istih oznaka. U sljedećem primjeru pohranjuju se dvije vrijednosti pod ključevima *foo* i *bar* i pri tome im je dodana oznaka *baz*:

```
<?php

Cache::tags(["baz"])->forever("foo", "qux");

Cache::tags(["baz"])->forever("bar", "quux");
```

Primjer koda 3.4. *Prikaz korištenja oznaka za podatke u pričuvnoj memoriji*

Budući da ključevi *foo* i *bar* sada imaju zajedničku oznaku moguće ih je istovremeno izbrisati na sljedeći način:

```
<?php

// vraća vrijednost "qux"
Cache::tags(["baz"])->get("foo");

// vraća vrijednost "quux"
Cache:: tags(["baz"])->get("bar");

Cache::tags(["baz"])->flush();

// vraća null
Cache::get("foo");

// vraća null
Cache::get("bar");
```

Primjer koda 3.5. *Prikaz dohvaćanja vrijednosti pomoću oznake i brisanje više vrijednosti pomoću jedne oznake*

3.6 Dohvaćanje i spremanje podataka za naslovnicu u pričuvnu memoriju

Najproblematičniji dio aplikacije je dohvaćanje podataka za naslovnu stranicu aplikacije. Naslovna stranica sastoji se od popisa objava (status ili fotografija) od svih prijatelja prijavljenog korisnika. Objava ima nekoliko dodatnih informacija, a to su:

- u slučaju da je objava status onda je potreban sadržaj statusa
- u slučaju da je objava fotografija onda je potrebna putanja do fotografije te (neobavezni) opis
- ime, prezime i ID korisnika koji je vlasnik objave
- svi komentari objava
- ukupan broj sviđanja na objavu

– informacija sviđa li se prijavljenom korisniku objava

Svi potrebni podaci spremaju se u MariaDB bazu podataka u nekoliko ključnih tablica. Ključne tablice su: *users*, *posts*, *statuses*, *timeline_photos*, *likes* i *comments*. Kako bi aplikacija radila što brže potrebno je optimizirati sve upite vezane za dohvaćanje podataka za naslovnu stranicu. Potrebno je izvesti 11 upita da bi se dohvatili svi potrebni podaci. Svaki od tih 11 upita je analiziran te optimiziran uvođenjem indeksa za ključne stupce koji se koriste u WHERE upitima. Time je značajno ubrzan proces dohvaćanja svih podataka s početnih oko 4 sekunde na oko 65 milisekundi. Iako su primjenom indeksa značajno smanjena vremena izvršavanja svakog upita, fokus ovoga rada nisu utjecaji indeksiranja tablica nego upotreba pričuvne memorije stoga se nije potrebno detaljnije osvrtati na performanse sa i bez indeksiranja. Budući da Laravel koristi apstrakciju za rukovođenje bazom podataka, SQL upiti se ne izvode direktno nego korištenjem ORM-a (eng. *object relational mapping*) zvanog Eloquent. U sljedećem primjeru prikazan je dio koda u kojem se dohvaćaju podaci za naslovnu stranicu:

```
<?php

// niz svih ID-eva koji su prijatelji korisnika
        $friendsIds = $user->getFriendsIds();

// niz ID-eva svih objava koje su se korisniku
svidjele$postsLikedByUser = \DB::table('likes')
     ->where('user_id', $user->id)
     ->pluck('post_id')
     ->toArray();

// upit za sve objave prijatelja sa svim dodatnim informacijama
$posts = Post::whereIn('user_id', $friendsIds)
     ->latest()
     ->forFeed();

// izvršavanje upita što uključuje i paginaciju
$posts = $posts->paginate(30, ['id', 'user_id', 'type_id',
     'created_at'], 'page', $page);

// dodatno procesiranje i formatiranje podataka potrebnih za naslovnu
stranicu
Post::appendToFeed($posts, $postsLikedByUser);

return $posts;
```

Primjer koda 3.6. *Dohvaćanje podataka za naslovnu stranicu*

Prethodni primjer prikazuje dohvaćanje podataka iz baze podataka te dodatno pripremanje podataka za prikaz na naslovnici. Međutim, prethodni primjer ne prikazuje pohranjivanje i dohvaćanje podataka iz pričuvne memorije. Potrebno je koristiti *remember* metodu klase *Cache*.

Ta metoda ima 3 parametra: naziv ključa, trajanje u minutama te *Closure* PHP funkciju. *Closure* funkcija se izvršava pri spremanju podataka u pričuvnu memoriju ukoliko nema vrijednosti pod tim ključem. Dakle, *remember* funkcija može čitati iz pričuvne memorije ukoliko postoji vrijednost pod tim ključem, a ako nema onda se izvršava *Closure* funkcija čija se povratna vrijednost sprema u pričuvnu memoriju.

```
<?php

$posts = \Cache::tags(["user-{$user->id}-feed"])
    ->remember("user-{$user->id}-feed", null, function () use ($user,
$page) {
        // niz svih ID-eva koji su prijatelji korisnika
        $friendsIds = $user->getFriendsIds();

        // niz ID-eva svih objava koje su se korisniku svidjele
        $postsLikedByUser = \DB::table('likes')
            ->where('user_id', $user->id)
            ->pluck('post_id')
            ->toArray();

        // upit za sve objave prijatelja sa svim dodatnim
informacijama
        $posts = Post::whereIn('user_id', $friendsIds)
            ->latest()
            ->forFeed();


        // izvršavanje upita što uključuje i paginaciju
        $posts = $posts->paginate(30, ['id', 'user_id', 'type_id',
'created_at'], 'page', $page);

        // dodatno procesiranje i formatiranje podataka potrebnih za
naslovnu stranicu
        Post::appendToFeed($posts, $postsLikedByUser);

        return $posts;
});

return $posts;
```

Primjer koda 3.7 *Dohvaćanje i spremanje podataka za naslovnu stranicu u Redis*

3.7 Korisnički podaci

Da bi se aplikacija mogla temeljito testirati potrebno je imati što više podataka. U tu svrhu generirani su lažni podaci korištenjem Faker PHP biblioteke. Faker biblioteka nudi razne funkcije kojima je moguće stvoriti mnogo vrsta podataka - email-ovi, rečenice, paragrafi, nasumični brojevi sa funkcijom distribucije vjerojatnosti, nasumično biranje elemenata iz postojećih nizova, *bool* vrijednosti itd. Korištenjem Faker biblioteke stvoreno je preko 1100 korisnika s jedinstvenom

elektroničkom adresom i lozinkom „secret“, preko 289 tisuća objava (od kojih su oko 229 tisuća statusi, a ostalih oko 51 tisuću fotografije). U bazi podataka pohranjeno je preko 92 tisuće podataka o prijateljstvima (uključuju aktivna i odbijena prijateljstva), oko 2,34 milijuna komentara te oko 7,39 milijuna sviđanja.

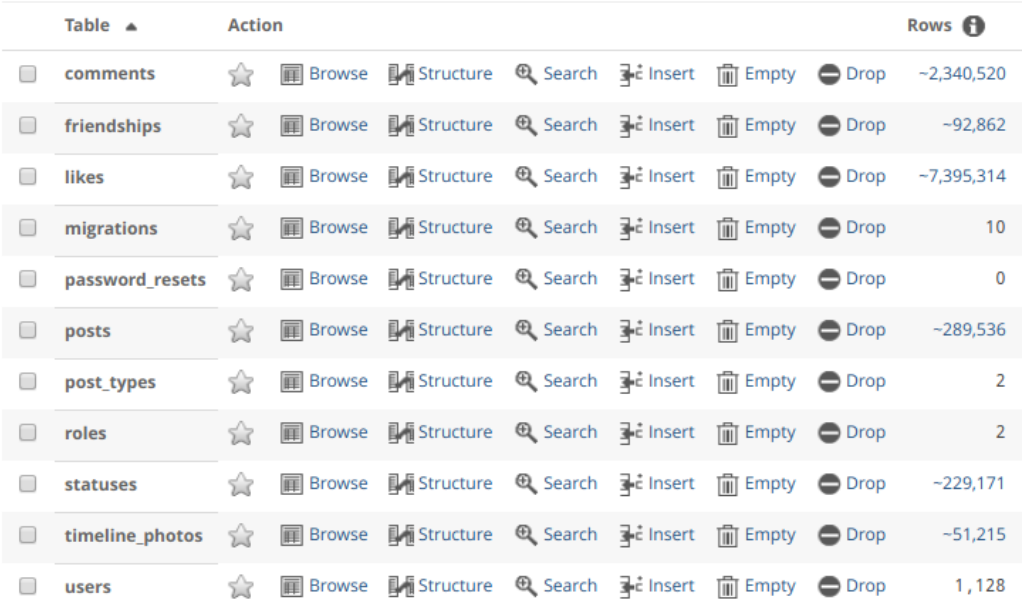


	Table	Action							Rows
☐	comments	☆	Browse	Structure	Search	Insert	Empty	Drop	~2,340,520
☐	friendships	☆	Browse	Structure	Search	Insert	Empty	Drop	~92,862
☐	likes	☆	Browse	Structure	Search	Insert	Empty	Drop	~7,395,314
☐	migrations	☆	Browse	Structure	Search	Insert	Empty	Drop	10
☐	password_resets	☆	Browse	Structure	Search	Insert	Empty	Drop	0
☐	posts	☆	Browse	Structure	Search	Insert	Empty	Drop	~289,536
☐	post_types	☆	Browse	Structure	Search	Insert	Empty	Drop	2
☐	roles	☆	Browse	Structure	Search	Insert	Empty	Drop	2
☐	statuses	☆	Browse	Structure	Search	Insert	Empty	Drop	~229,171
☐	timeline_photos	☆	Browse	Structure	Search	Insert	Empty	Drop	~51,215
☐	users	☆	Browse	Structure	Search	Insert	Empty	Drop	1,128

Sl. 3.2 *Prikaz tablica iz baze podataka te broj redaka u svakoj tablici*

4. IZVOĐENJE TESTOVA

4.1 Ciljevi testiranja

Prvi cilj testiranja je pronaći maksimalni broj obrađenih HTTP zahtjeva po sekundi do pojave prve pogreške dvije varijante aplikacije - sa i bez upotrebe Redisa. Najpouzdanija metoda za utvrđivanje broja maksimalnog broja zahtjeva po sekundi je opterećivanje poslužitelja, tzv. *stress testing*.

Drugi cilj testiranja je utvrditi prosječno vrijeme odziva naslovne stranice tijekom inicijalnog opterećivanja, ali i u drugim scenarijima gdje nije cilj doći do prve pogreške nego opteretiti server do kritične razine prije prve pogreške. Ta razina utvrđuje se empirijski - isprobavanjem različitih parametara te postupnom prilagodbom prema određenim vrijednostima.

Najtočnije opterećivanje trebalo bi biti izvedeno s pravim korisnicima koji izvode prave radnje u aplikaciji. Međutim, organizirati takvu vrstu testiranja s velikim brojem korisnika gotovo je nemoguće u stvarnosti. Ako je potrebno testirati primjenu neke nove tehnike ili tehnologije u trenutačnoj aplikaciji, onda je najbolje testirati tu primjenu na određenom podskupu korisnika i temeljito bilježiti sve ključne informacije. U aplikacijama koje još uvijek nemaju velik promet ili imaju povremeno velik promet, testiranje je najbolje odraditi koristeći posebne alate za simulaciju prometa većeg broja korisnika. Takvim testiranjima moguće je saznati više detalja o performansama aplikacije te utvrditi slabe točke. Jedan od popularnih softvera za kompleksniju simulaciju većeg broja korisnika naziva se Apache JMeter.

4.2 Programska podrška za testiranje

Apache JMeter je program otvorenog koda dizajniran za kreiranje funkcijskih testova u velikom broju te mjerenje performansi mrežnih aplikacija. Program je napisan u Java programskom jeziku te je dostupan na nekoliko platformi. Podržava opterećenje raznih vrsta aplikacija, poslužitelja i protokola kao što su HTTP, SMTP, POP3, TCP, FTP itd. JMeter se koristi u dva osnovna načina rada: kao program s grafičkim korisničkim sučeljem (eng. *graphical user interface* - GUI) te kao program u komandnoj liniji [12]. GUI način rada preporuča se samo za kreiranje testnih planova dok se za sam proces opterećivanja preporučuje korištenje komandne linije radi bolje optimizacije.

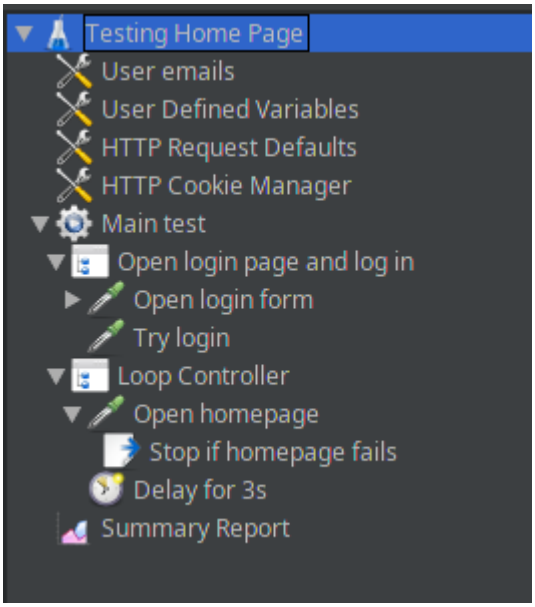
JMeter omogućuje simulaciju većeg prometa tako što istovremeno simulira ponašanje željenog broja korisnika. Moguće je definirati točan broj korisnika i točan broj njihovih radnji kroz određen

period. U terminologiji JMetera korisnik se naziva *thread* (hrv. nit), a jedna korisnikova radnja naziva se *sample* (hrv. uzorak) [12]. Budući da se radi o testiranju HTTP protokola, radi jednostavnosti će se u nastavku teksta koristiti pojmovi korisnik umjesto nit te radnja umjesto uzorak. Testiranje pomoću JMeter programa svodi se na definiranje testnog plana, postavljanje osnovnih parametara te izvođenje testa. Definiranje plana podrazumijeva definiranje svih radnji testa koristeći JMeter komponente. Postoji nekoliko vrsti komponenata s kojima se na vizualni način može programirati tijek testa. Primjerice, moguće je definirati test koji se sastoji od odlaska na stranicu za prijavu, pokušaja prijave unošenjem potrebnih podataka i pritiskom na određen gumb, te interpretiranje povratnog HTTP koda ili samog sadržaja nakon pokušaja prijave. Sami podaci za prijavu se definiraju u konfiguraciji komponenti ili se dinamički uzimaju iz nekog drugog izvora, npr. CSV datoteka koja se sastoji od parova elektroničkih adresi i lozinki. JMeter ima velik broj komponenti koje najčešće imaju nekoliko konfiguracijskih opcija. Budući da za razumijevanje ovog rada nije potrebno detaljno poznavanje JMeter alata, pojedinosti korištenih komponenti neće biti detaljno opisane.

4.3 Definiranje testnog plana i podataka

Najvažnija i najkompleksnija funkcionalnost testne društvene mreže je prikaz naslovnice stoga je ona u fokusu testiranja. Osnovni testni plan se sastoji od sljedećih radnji korisnika:

1. Otvaranje stranice za prijavu
2. Unos potrebnih podataka za prijavu te pokušaj prijave
3. Povremeno otvaranje naslovne stranice (u prosjeku svake 4 sekunde sa standardnom devijacijom funkcije normalne distribucije od 500 milisekundi)



Sl. 4.1 *Prikaz gotovog testnog plana u JMeter programu*

Test počinje sa manjim brojem korisnika te se povremeno dodaje sve veći broj korisnika da bi sustav dobivao postupno opterećenje čime se daje više prostora za optimalno korištenje pričuvne memorije svih ostalog programa uključenih u obradu HTTP zahtjeva (npr. HTTP poslužitelj, PHP interpreter, MySQL itd.).

4.4 Izvedba testa opterećenja do prve pogreške

Za izvedbu opterećenja do prve pogreške potrebno je zadati preveliki broj korisnika u relativno kratkom vremenskom okviru. Parametar broj korisnika postavljen je na 500, a kao vrijeme uspona broja korisnika postavljeno je 600 sekundi. Vrijeme uspona označava period u kojem će broj korisnika rasti od 0 do postavljenog broja korisnika, u ovome slučaju 500. Radi najpouzdanijih performansi testiranje je provedeno korištenjem načina rada u komandnoj liniji, a testni program pokrenut je na osobnom računalu koje je posebno konfigurirano da podržava velik broj aktivnih HTTP zahtjeva. Testirane su dvije varijante aplikacije, sa i bez Redisa kao pričuvna memorija.

Tab. 4.1 *Parametri testa opterećivanja do prve pogreške*

Parametar	**Vrijednost**
Broj korisnika	500
Vrijeme uspona [s]	600

Tab. 4.2 *Rezultati testa opterećenja do prve pogreške*

Mjereni atribut	**Bez Redisa**	**S Redisom**	**Razlika [%]**
Prohodnost [br. zahtj./sek]	14,53	17.23	18,58%
Prosječno vrijeme odziva [ms]	10291,04	7680,08	33,99%
Broj radnji	7644	8635	12,96%
Vrijeme trajanja do prve pogreške [s]	509	488	4,3%

Rezultati testiranja opterećenjem do prve pogreške pokazuju da korištenje Redisa smanjuje vrijeme odziva za 33,99% što značajno poboljšava iskustvo krajnjeg korisnika. Prohodnost zahtjeva (prosječni broj zahtjeva po sekundi) je veća za 18,58% što znači da aplikacija može podnijeti veći promet istovremeno. Varijanta bez Redisa je neočekivano uspjela izdržati dulji period do prve pogreške. Razlog tomu je veće korištenje procesora jer spremanje i čitanje podataka iz Redisa zahtjeva dodatnu serijalizaciju i deserijalizaciju što postane problematično u određenoj točki zasićenja.

4.5 Izvedba testa neprekidnog opterećenja bez pogrešaka kroz određeni period

Cilj ovoga testa je utvrditi vrijeme odziva i prohodnost pri velikom opterećenju pri kojem ne dolazi do greške. Postavljeni parametri utvrđeni su empirijskim testiranjem tako što su se postupno prilagođavali parametri testa koji utječu na opterećenje sve dok se test nije uspio izvesti u cijelosti bez stranice pogreške. U ovome testu broj korisnika je postavljen na 250, a vrijeme uspona na 500 sekundi. Test ima fiksno vrijeme trajanja od 720 sekundi (12 minuta).

Tab. 4.3 *Parametri testa neprekidnog opterećenja bez pogreške kroz određeni period*

Parametar	**Vrijednost**
Broj korisnika	250
Vrijeme uspona [s]	500
Vrijeme trajanja [s]	720

Tab. 4.4 *Rezultati neprekidnog testa opterećenja kroz određeni period bez pogreške*

Mjereni atribut	**Bez Redisa**	**S Redisom**	**Razlika [%]**
Prohodnost [br. zahtj./sek]	16,14	18,24	13,01%
Prosječno vrijeme odziva [ms]	6021,84	4897,40	22,95%
Broj radnji	11767	13258	12,67%

Rezultati ovoga testa su očekivano konzistentni s prethodnim testom. Varijanta s Redisom ima brže prosječno vrijeme odziva, prohodnost i ukupan broj radnji.

5. ZAKLJUČAK

Kao što je vidljivo po rezultatima testova, glavna prednost korištenja Redisa kao pričuvne memorije je poboljšanje performansi aplikacije. Precizno poboljšanje performansi je iznimno teško izračunati jer ovisi o broju SQL upita, načinu korištenja stranice, količini podataka koja se mora spremiti u pričuvnu memoriju, broj aktivnih korisnika itd. Na primjeru naslovne stranice jednostavne društvene mreže prohodnost je veća za oko 15% i vrijeme odziva pri opterećenju je barem 20% manje. Na skali velike društvene mreže ovim pristupom može se značajno uštedjeti na sklopovlju. Što je proces dohvaćanja podataka iz primarne baze podataka kompliciraniji i što je broj upita veći, to je značajnije poboljšanje stabilnosti i performansi. Primjerice, za stranice kojima je potrebno nekoliko desetaka ili čak više od 100 SQL upita, Redis može donijeti nekoliko puta brže odzive i prohodnost. Postoje neke vrste aplikacija koje se isključivo oslanjaju na Redis za čitanje podataka kao npr. Magento *e-commerce* radni okvir.

Međutim, postoji nekoliko mana zbog kojih je implementiranje Redisa kao pričuvnu memoriju relativno težak zadatak. Prije svega je potrebno u svoju postojeću infrastrukturu instalirati i konfigurirati Redis server i klijent. To može biti problem ukoliko je infrastruktura velika i troma. Takav pothvat zahtjeva nezanemarivu količinu rada administratora sustava. Još veći problem je općeniti problem vezan uz pričuvnu memoriju, a to je integritet i ažurnost podataka u pričuvnoj memoriji. Što su podaci promjenjiviji to je problem veći i teži za riješiti, te su najčešće dobici na performansama manji. Za osiguranje integriteta korištenih podataka potrebno je implementirati dodatnu logiku što zahtjeva više koda, programerskih sati. Potencijalne posljedice su neispravnost (neažurnost) prikazanih podataka te druge povezane pogreške. Primjerice, u kontekstu testne aplikacije, ako spremimo naslovnicu korisnika u pričuvnu memoriju i ukoliko netko od prijatelja korisnika nešto komentira ili sviđa, potrebno je ili osvježiti te podatke u pričuvnoj memoriji (zamjena starih podataka s novim), ili jednostavno izbrisati te podatke u potpunosti što jamči integritet podataka pri sljedećem zahtjevu (bit će direktno dohvaćeni iz primarne baze podataka).

LITERATURA

[1] „LEMP Stack Info | Configure Linux, NGINX, MySQL, & PHP“, *LEMP Stack Info*. [Na internetu]. Dostupno na: https://lemp.io/. [Pristupljeno: 15.09.2018].

[2] „NGINX | High Performance Load Balancer, Web Server, & Reverse Proxy“, *NGINX*. [Na internetu]. Dostupno na: https://www.nginx.com/. [Pristupljeno: 15.09.2018].

[3] „MariaDB“, *Wikipedia*. [Na internetu]. Dostupno na: https://en.wikipedia.org/w/index.php?title=MariaDB&oldid=859343289. [Pristupljeno: 15.09.2018].

[4] „PHP: History of PHP - Manual“. [Na internetu]. Dostupno na: http://php.net/manual/en/history.php.php. [Pristupljeno: 15.09.2018].

[5] L. Ullman, *PHP and MySQL for Dynamic Web Sites: Visual QuickPro Guide*, 5. izdanje. Berkeley, California: Peachpit Press, 2017.

[6] „Understanding Model-View-Controller“. [Na internetu]. Dostupno na: https://blog.codinghorror.com/understanding-model-view-controller/. [Pristupljeno: 15.09.2018].

[7] „Redis“. [Na internetu]. Dostupno na: https://redis.io/. [Pristupljeno: 15.09.2018].

[8] „How can I connect to redis using php but without use a client library“, *Stack Overflow*. [Na internetu]. Dostupno na: https://stackoverflow.com/questions/9641347/how-can-i-connect-to-redis-using-php-but-without-use-a-client-library/9641796. [Pristupljeno: 15.09.2018].

[9] D. Alessandri, *Flexible and feature-complete Redis client for PHP and HHVM: nrk/predis*. 2018.

[10] „Mail - Laravel - The PHP Framework For Web Artisans“. [Na internetu]. Dostupno na: https://laravel.com/docs/5.6/mail#driver-prerequisites. [Pristupljeno: 15.09.2018].

[11] „Cache - Laravel - The PHP Framework For Web Artisans“. [Na internetu]. Dostupno na: https://laravel.com/docs/5.6/cache#driver-prerequisites. [Pristupljeno: 15.09.2018].

[12] „Apache JMeter - Apache JMeter™“. [Na internetu]. Dostupno na: https://jmeter.apache.org/. [Pristupljeno: 15.09.2018].

SAŽETAK

Ovaj rad opisuje korištenje Redis baze podataka kao pričuvne memorije web aplikacije te utjecaj Redisa na performanse aplikacije. Cilj je demonstrirati kako koristiti Redis u aplikaciji baziranoj na LEMP stogu. Za potrebe rada izrađena je jednostavna društvena mreža zvana Feritbook. Feritbook je napravljen koristeći Laravel PHP radni okvir. U radu je prikazano kako koristiti Redis u PHP-u bez biblioteke, s bibliotekom Predis te koristeći Laravel apstrakcije za upravljanje pričuvnom memorijom koristeći *Cache* klasu. Demonstrirano je kako spremati i dohvaćati podatke za naslovnu stranicu u pričuvnu memoriju. Za testiranje performansi korišten je testni alat Apache JMeter. Koristeći JMeter, izvedene su dvije simulacije koje mjere bitne značajke performansi kod dvije varijante aplikacije – sa i bez Redisa. Dobivenim rezultatima utvrđena su poboljšanja performansi Redis varijante pod velikim opterećenjem. Međutim, također je utvrđeno da se korištenjem Redisa povećava kompleksnost *backend* arhitekture i samog koda što može dovesti do neželjenih pogrešaka i neažurnosti podataka.

Ključne riječi: LEMP, performanse, pričuvna memorija, Redis, web aplikacija

ABSTRACT

Advantages and disadvantages of Redis technology in web applications

This graduate paper deals with using Redis database technology as cache memory in web applications. The goal of this paper is to demonstrate how to use Redis in an application that is based on LEMP stack. A social network application called Feritbook was built for the purposes of this demonstration. Feritbook is built using Laravel PHP framework. The paper displays how to use Redis in PHP without library, with Predis library, and by using Laravel cache abstraction layer for managing cache – *Cache* class. It is demonstrated how to set and get data to and from Redis cache database. A testing tool called Apache JMeter was used for the performance testing. Two simulations that measure significant features with two variants of the application were made by using JMeter – first with and the second time without Redis. The results confirmed that Redis variant performs better under overload. However, it was also confirmed that using Redis increases the complexity of the backend architecture and the code itself which can lead to unwanted errors and out-of-date data.

Key words: LEMP, performance, cache memory, Redis, web application

ŽIVOTOPIS

Ivan Bernatović rođen je u Slavonskom Brodu 16. srpnja 1993. godine. Osnovnu školu završio je u Županji u razdoblju od 2000. do 2008. godine. Upisuje opću gimnaziju u Županji 2008. godine, koji završava 2012. godine. Nakon toga upisuje Elektrotehnički fakultet u Osijeku, preddiplomski studij, smjer računarstvo kojeg završava 2015. godine. 2016. godine upisuje diplomski studij na prethodno spomenutom fakultetu, sada Fakultet elektrotehnike, računarstva i informacijskih tehnologija, smjer računarstvo, izborni blok programsko inženjerstvo kojega trenutačno pohađa. Tijekom studija je profesionalno radio kao backend web developer u tvrtki Inchoo. Trenutno je samozaposlen kao nezavisni web developer čime se bavi 2 godine.

Potpis

_____

PRILOG

Na optičkom disku priloženom uz ovaj diplomski rad nalaze se .docx i .pdf verzija diplomskog rada te izvorni kod testne aplikacije.